走近Node.js的异步代码设计

许多企业目前在评估Node.js的异步、事件驱动型的I/O，认为这是一种高性能方案，可以替代多线程企业应用服务器的传统同步I/O。异步性质意味着，企业开发人员必须学习新的编程模式，忘掉旧的编程模式。他们必须彻底转变思路，可能需要借助电击疗法^_^。本文介绍了如何将旧的同步编程模式换成全新的异步编程模式。

作者：布加迪编译 来源：51CTO | 2011-12-23 13:58

【51CTO精选译文】许多企业目前在评估Node.js的异步、事件驱动型的I/O，认为这是一种高性能方案，可以替代多线程企业应用服务器的传统同步I/O。异步性质意味着，企业开发人员必须学习新的编程模式，忘掉旧的编程模式。他们必须彻底转变思路，可能需要借助电击疗法^_^。本文介绍了如何将旧的同步编程模式换成全新的异步编程模式。

51CTO推荐专题：Node.js专区

开始转变思路

编辑推荐

热点 惹毛程序员的十件事！需求变更居然不是排第一！

头条 高性能Java持久化的14个技巧

要使用Node.js，就有必要了解异步编程的工作原理。异步代码设计并非简单的设计，需要一番学习。现在需要来一番电击疗法：本文在同步代码示例旁边给出了异步代码示例，表明如何更改同步代码，才能变成异步代码。这些示例都围绕Node.js的文件系统(fs)模块，因为它是唯一含有同步I/O操作及异步I/O操作的模块。有了这两种示例，你可以开始转变思路了。

相关代码和独立代码

回调函数(callback function)是Node.js中异步事件驱动型编程的基本构建模块。它们是作为变量，传递给异步I/O操作的函数。一旦操作完成，回调函数就被调用。回调函数是Node.js中实现事件的机制。

下面显示的示例表明了如何将同步I/O操作转换成异步I/O操作，并显示了回调函数的使用。示例使用异步fs.readdirSync()调用，读取当前目录的文件名称，然后把文件名称记录到控制台，最后读取当前进程的进程编号(process id)。

同步

```
var fs = require('fs'),
    filenames,
    i,
    processId;
filenames = fs.readdirSync(".");
for (i = 0; i < filenames.length; i++) {
    console.log(filenames[i]);
}
console.log("Ready.");
processprocessId = process.getuid();
```

异步

```
1.  var fs = require('fs'),
2.      processId;
3.  fs.readdir(".", function (err, filenames) {
4.      var i;
5.      for (i = 0; i < filenames.length; i++) {
6.          console.log(filenames[i]);
7.      }
8.      console.log("Ready.");
9.  });
10. processprocessId = process.getuid();
```

在同步示例中，处理器等待fs.readdirSync() I/O操作，所以这是需要更改的操作。Node.js中该函数的异步版本是fs.readdir()。它与fs.readdirSync()一样，但是回调函数作为第二个参数。

使用回调函数模式的规则如下：把同步函数换成对应的异步函数，然后把原先在同步调用后执行的代码放在回调函数里面。回调函数中的代码与同步示例中的代码执行一模一样的操作。它把文件名称记录到控制台。它在异步I/O操作返回之后执行。

就像文件名称的记录依赖fs.readdirSync() I/O操作的结果，所列文件数量的记录也依赖其结果。进程编号的存储独立于I/O操作的结果。因而，必须把它们移到异步代码中的不同位置。

规则就是将相关代码移到回调函数中，而独立代码的位置不用管。一旦I/O操作完成，相关代码就被执行，而独立代码在I/O操作被调用之后立即执行。

顺序

同步代码中的标准模式是线性顺序：几行代码都必须下一行接上一行来执行，因为每一行代码依赖上一行代码的结果。在下面示例中，代码首先变更了文件的访问模式(比如Unix chmod命令)，对文件更名，然后检查更名后文件是不是符号链接。很显然，该代码无法乱序运行，不然文件在模式变更前就被更名了，或者符号链接检查在文件被更名前就执行了。这两种情况都会导致出错。因而，顺序必须予以保留。

同步

```
var fs = require('fs'),
    oldFilename,
    newFilename,
    isSymLink;
oldFilename = "./processId.txt";
newFilename = "./processIdOld.txt";
fs.chmodSync(oldFilename, 777);
fs.renameSync(oldFilename, newFilename);
isSymLink = fs.lstatSync(newFilename).isSymbolicLink();
```

异步

```
var fs = require('fs'),
    oldFilename,
    newFilename;
oldFilename = "./processId.txt";
newFilename = "./processIdOld.txt";
fs.chmod(oldFilename, 777, function (err) {
    fs.rename(oldFilename, newFilename, function (err) {
        fs.lstat(newFilename, function (err, stats) {
            var isSymLink = stats.isSymbolicLink();
        });
```

```
    });
});
```

在异步代码中，这些顺序变成了嵌套回调。该示例显示了fs.lstat()回调嵌套在fs.rename()回调里面，而fs.rename()回调嵌套在fs.chmod()回调里面。

并行处理

异步代码特别适合操作I/O操作的并行处理：代码的执行并不因I/O调用的返回而受阻。多个I/O操作可以并行开始。在下面示例中，某个目录中所有文件的大小都在循环中累加，以获得那些文件占用的总字节数。使用异步代码，循环的每次迭代都必须等到获取单个文件大小的I/O调用返回为止。

异步代码允许快速连续地在循环中开始所有I/O调用，不用等结果返回。只要其中一个I/O操作完成，回调函数就被调用，而该文件的大小就可以添加到总字节数中。

唯一必不可少的有一个恰当的停止标准，它决定着我们完成处理后，就计算所有文件的总字节数。

同步

```
var fs = require('fs');
function calculateByteSize() {
    var totalBytes = 0,
        i,
        filenames,
        stats;
    filenames = fs.readdirSync(".");
    for (i = 0; i < filenames.length; i ++) {
```

```
        stats = fs.statSync("./" + filenames[i]);
        totalBytes += stats.size;
    }
    console.log(totalBytes);
}



calculateByteSize();
```

异步

```
var fs = require('fs');
var count = 0,
    totalBytes = 0;
function calculateByteSize() {
    fs.readdir(".", function (err, filenames) {
        var i;
        count = filenames.length;
        for (i = 0; i < filenames.length; i++) {
            fs.stat("./" + filenames[i], function (err, stats) {
                totalBytes += stats.size;
                count--;
                if (count === 0) {
                    console.log(totalBytes);
                }
            });
        }
    });
}
calculateByteSize();
```

同步示例简单又直观。在异步版本中，第一个fs.readdir()被调用，以读取目录中的文件名称。在回调函数中，针对每个文件调用fs.stat()，返回该文件的统计信息。这部分不出所料。

值得关注的方面出现在计算总字节数的fs.stat()回调函数中。所用的停止标准是目录的文件数量。变量count以文件数量来初始化，倒计数回调函数执行的次数。一旦数量为0，所有I/O操作都被回调，所有文件的总字节数被计算出来。计算完毕后，字节数可以记录到控制台。

异步示例有另一个值得关注的特性：它使用闭包(closure)。闭包是函数里面的函数，内层函数访问外层函数中声明的变量，即便在外层函数已完成之后。fs.stat()回调函数是闭包，因为它早在fs.readdir()回调函数完成后，访问在该函数中声明的count和totalBytes这两个变量。闭包有关于它自己的上下文。在该上下文中，可以放置在函数中访问的变量。

要是没有闭包，count和totalBytes这两个变量都必须是全局变量。这是由于fs.stat()回调函数没有放置变量的任何上下文。calculateBiteSize()函数早已结束，只有全局上下文仍在那里。这时候闭包就能派得上用场。变量可以放在该上下文中，那样可以从函数里面访问它们。

代码复用

代码片段可以在JavaScript中复用，只要把代码片段包在函数里面。然后，可以从程序中的不同位置调用这些函数。如果函数中使用了I/O操作，那么改成异步代码时，就需要某种重构。

下面的异步示例显示了返回某个目录中文件数量的函数countFiles()。countFiles()使用I/O操作 fs.readdirSync() 来确定文件数量。 span style="font-family: courier new,courier;">countFiles()本身被调用，使用两个不同的输入参数：

同步

```
var fs = require('fs');
var path1 = "./",
    path2 = "../../";
function countFiles(path) {
    var filenames = fs.readdirSync(path);
    return filenames.length;
}
console.log(countFiles(path1) + " files in " + path1);
console.log(countFiles(path2) + " files in " + path2);
```

异步

```
var fs = require('fs');
var path1 = "./",
    path2 = "../../",
    logCount;
function countFiles(path, callback) {
    fs.readdir(path, function (err, filenames) {
        callback(err, path, filenames.length);
    });
}
logCount = function (err, path, count) {
    console.log(count + " files in " + path);
};
countFiles(path1, logCount);
countFiles(path2, logCount);
```

把fs.readdirSync()换成异步fs.readdir()迫使闭包函数cntFiles()也变成异步，因为调用cntFiles()的代码依赖该函数的结果。毕竟，只有fs.readdir()返回后，结果才会出现。这导致了cntFiles()重构，以便还能接受回调函数。整个控制流程突然倒过来了：不是console.log()调用cntFiles()，cntFiles()再调用fs.readdirSync()，在异步示例中，而是cntFiles()调用fs.readdir()，然后cntFiles()再调用console.log()。

结束语

本文着重介绍了异步编程的一些基本模式。将思路转变到异步编程绝非易事，需要一段时间来适应。虽然难度增加了，但是获得的回报是显著提高了并发性。结合JavaScript的快速周转和易于使用等优点，Node.js中的异步编程有望在企业应用市场取得进展，尤其是在新一代高度并发性的Web 2.0应用程序方面。

原 文 ： http://shinetech.com/thoughts/thought-articles/139-asynchronous-code-design-with-nodejs

【编辑推荐】

1. 使用Node.js开发多人玩的HTML 5游戏
2. Node.js提速指南
3. Node.js专区
4. 什么是Node.js？
5. 使用node.js进行服务器端JavaScript编程

【责任编辑：陈贻新 TEL：（010）68476606】

点赞 0

node.js 分享:

内容点评 已有 0 条评论, 0 次赞

还可以输入500字

请输入你的评论

您还没有登录！请先 登录 或 注册

提交

还没有评论内容

大家都在看 猜你喜欢

新浪微博胡南炜：深度学习在微博信息流推荐中的实践

甲骨文云平台全面升级，助力客户稳步上云

迅雷战略转型，开创“共享计算”时代

微软技术暨生态大会，纳德拉为小冰点赞